MPDG - Mobile Profile based Distributed Grouping

Christian Seitz, Michael Berger
Siemens AG, Corporate Technology, Information and Communications
D-81730 Munich, Germany
christian.seitz@mchp.siemens.de, m.berger@siemens.com

Bernhard Bauer
Institute of Computer Science, University of Augsburg
86150 Augsburg, Germany
bernhard.bauer@informatik.uni-augsburg.de

Abstract

We present a new kind of mobile ad hoc application, which we call Mobile Profile based Distributed Grouping (MPDG), which is a combination of mobile clustering and data clustering. In MPDG each mobile host is endowed with a user profile and while the users move around, hosts with similar profiles are to be found and a robust mobile group is formed. The members of a group are able to cooperate or attain a goal together.
In this paper MPDG is defined and it is compared with related approaches. Furthermore, a modular architecture and algorithms are presented to build arbitrary MPDG applications.

1. Introduction

Tomorrow's world will be intrinsically ubiquitous and mobile. Ubiquitous computing is a new trend in computation and communication. It is an intersection of several technologies, including embedded devices, service discovery, wireless networking and personal computing technologies. In an ad hoc network, mobile devices can detach completely from the fixed infrastructure and establish transient and opportunistic connections with other devices that are in communication range. The structure of an ad hoc mobile network could be highly dynamic.
We present a new mobile ad hoc network application area, which we call *Mobile Profile based Distributed Grouping* (MPDG). In MPDG each mobile host is endowed with a *user profile*. A *user profile* (short: profile) is a comprehensive data collection belonging to a specific object (e. g. a person). A profile consists of a set of parameters defining the configuration of a user specific application. While the users move around, mobile hosts with similar profiles are to be found and a mobile group is formed. The participants of a group are able to cooperate or attain a goal together.
The paper is organized as follows. Section 2 gives an overview of related work of other clustering or grouping problems. The next section presents the architecture of a MPDG application. Section 4 describes the used algorithms and presents simulation results. Finally, section 5 concludes the paper with a summary.

2. Problem Classification and Related Work

In this section we classify with which problems a *Mobile Profile based Distributed Grouping* application is confronted. Furthermore, we show which other research areas are related and how this new problem has already been discussed in literature.

2.1. Problem Classification

In *Mobile Profile based Distributed Grouping* mobile hosts are equipped with wireless transmitters, receivers, and a user profile. They are moving in a geographical area and are forming an ad hoc network. In this environment hosts with similar profiles have to be found. *Mobile Profile based Distributed Grouping* in ad hoc environments comprises three main problems which have to be solved to accomplish a MPDG application. The first problem is the dynamic behavior of an ad hoc network, where the number of mobile hosts and communication links permanently changes. Secondly, a data structure for the user profile has to be defined and a mechanism must be created to compare profile instances. Finally, similar profiles have to be found in the ad hoc network and the corresponding host form a group.

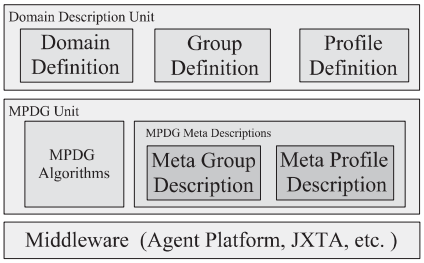

Figure 1. Components of a MPDG application

2.2. Related Research Areas

Grouping algorithms and their applications appear very often in literature. There are mainly two different research areas associated with it, namely mobile networks, databases and data mining. Grouping in mobile networks describes the partitioning of a mobile network in several, mostly disjoint, clusters [1, 2]. This clustering takes place at the network layer and is used for routing purposes.
Clustering is also known in the database or data mining area. A huge amount of data is scanned with the goal to find similar data sets. This research is also known as unsupervised learning. In the surveys of Fasulo [4] or Fraley and Raftery [5] an overview of many algorithms for that domain can be found.
MPDG combines the two aforementioned clustering approaches. The problems, arising by means of the motion of the hosts could be solved by methods used in the mobile network area. Searching for similar profiles is based on algorithms of data clustering. Both methods must be adapted to MPDG, e. g. while in the database area millions of data sets must be scanned, in the MPDG application at the utmost one hundred other hosts are present. In contrast to data sets in databases ad hoc hosts move around and are active, i. e. they can publish their profile by their own.

3. MPDG Application Architecture

In this section the architecture of a MPDG application is presented. Finally, some assumptions to the application are made.

3.1. Architecture

The architecture of MPDC applications is depicted in figure 1. A MPDG application consists of three essential parts: the middleware, the MPDG unit, and the Domain Description unit.
The middleware establishes the basis for a MPDG application. It is in charge of detecting other hosts in the mobile environment and provides a mechanism for sending and receiving messages to other hosts, which are within transmission range. The central element of a MPDG application is the MPDG unit. It is made up of a MPDG algorithm entity and a MPDG Description entity.
As the MPDG unit is totally domain independent, the structure of a profile or a group definition must be defined. This is done by the *Meta Profile Description* and the *Meta Group Description* which are elements of the *MPDG Description entity*. The Meta Group and Meta Profile Description define the content of a group or profile definition and specify optional and mandatory elements of a group or profile definition. In both meta descriptions, there is a mandatory general part, defining the name of the profile or group. The meta profile description encompasses a set of tags for profile entry definitions, and specifies the structure of rules, that can be declared in the profile definition in the Domain Description Unit. The meta group description comprises abstract tag definitions for the size of a group, the profile elements a group consists of, and the properties to become a group member.
On top of the MPDG unit, the *Domain Description* unit is located. This unit adjusts the MPDG unit to a specific application domain. In the *Domain Definition* entity, domain dependent knowledge is described and in the profile and group definition entities the domain specific profiles and group properties are defined. The *Profile Definition* specifies the structure of the profile for the domain, in accordance with the Meta Profile description.
The *Group* term varies from application to application. A group can be a few people with similar properties (the same hobby, profession, age etc.) but also a set of machines with totally different capabilities. For that reason the *Group Definition* comprises the characteristics of the group ought to be found.
In this paper, we concentrate on describing the MPDG algorithm entity (see section 4).

3.2. Assumptions

In the following the assumptions with which we are currently working are mentioned.

- We do not rely on any central component.
- There is no location information available, e. g. GPS-data or cell-info.
- The transmission range of all hosts is r_t.

4. MPDG Algorithms

In this section the architecture of the algorithm entity is presented, the used network model is defined and the algorithms for each layer are shown. The algorithm entity has a

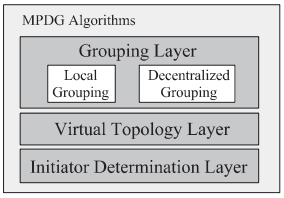

Figure 2. MPDG Algorithm Entity

layered architecture and encompasses algorithms for initiator determination, virtual topology creation, local grouping and decentralized grouping. Finally, we show some simulation results, in order to indicate how stable the generated groups are.

4.1. MPDG Algorithm Entity

The most important part of a MPDG application is the Algorithm Entity (AE). The design of this essential entity is shown in figure 2. The lowest layer of the AE is the *Initiator Detection Layer*, which assigns the initiator role to some hosts. Initiators are needed in order to guarantee, that the algorithm of the next layer is not started by each host of the network. This layer does not determine one single initiator for the whole ad hoc network. It is sufficient, if the number of initiator nodes is only reduced.
The *Virtual Topology Layer* is responsible for covering a given ad hoc topology with another topology, e. g. a tree or a logical ring. This virtual topology is necessary to reduce the number of messages, that are sent by the mobile hosts while the grouping process. First experiences show, that a tree is the most suitable virtual topology and therefore we will only address the tree approach in this paper.
The next layer, the *Grouping Layer*, is the most important one. The MPDG algorithms entity distinguishes between *Local Grouping* and *Global Grouping*. In order to obtain a decentralized group a two-tiered process is started. At first, each host selects from its neighbor hosts, the subset of similar hosts which ought to be in the group. This set of hosts is called a *Local Group*. After each host has determined its local group, these groups are exchanged in a second step and a unique *Global group* is acquired.
In the next subsections each layer is described in detail.

4.2. Initiator Determination

Before the spanning tree is created, the initiators must be determined who are allowed to send the first `creation`-messages. Without initiators all hosts start randomly sending messages with the result that a tree will never be created. We are not in search of one single initiator, we only want to guarantee, that not all hosts start the initiation.

There are two ways to determine the initiator, an active and a passive one. The active approach starts an election algorithm (see Malpani *et al.* [7]). These algorithms are rather complex, i. e. a lot of messages are sent which is very time consuming. They guarantee that only one initiator is elected and in case of link failures that another host takes the initiator role. Such a procedure is not appropriate and not necessary for MPDG, because the initiator is only needed once and it matters little if more than one initiator is present. Therefore, we decided for the passive determination method, which is similar to Gafni and Bertsekas [6]. By applying the passive method no message is sent in the beginning to determine an initiator. Each host has an ID and knows all neighbor IDs. The IDs are provided by the the middleware when mobile devices come into transmission range. We only allow a host being an initiator, if its ID is larger than all IDs of its neighbors. The initiator is in charge of starting the virtual topology algorithm, described in the next section.

4.3. Virtual Topology Creation

Having confined the number of initiators, a ad hoc network can be covered with a virtual topology (VT) with the algorithm described below. Simulations showed that a *spanning tree* is a promising approach for a VT and therefore we will only describe the spanning tree VT in this paper.

The Algorithm
Each host keeps a spanning tree sender list (STSL). The STSL contains the subset of a host's neighbors belonging to the spanning tree. The initiator, determined in the previous section, sends a `create`-message furnished with its ID to all its neighbors. If a neighbor receives a `create`-message for the first time, this message is forwarded to all neighbors except for the sender of the `create`-message. The host adds each receiver to the STSL. If a host receives a message from a host which is already in the STSL, it is removed from the list.
To identify a tree, the ID of the initiator is always added to each message. It may occur that a host already belongs to another tree. Under these circumstances the message is not forwarded any more and the corresponding host belongs to two (more are also possible) trees. The host decides to a tree later, while the grouping process.
In order to limit the tree size a hop-counter c_h is enclosed to each message and is each time decremented, the message is forwarded. If the counter is equal to zero, the forwarding process stops. Note, with an increasing c_h the time for building a group also increases, because c_h is equivalent to the half diameter d of a ad hoc network.
By using a hop-counter it may occur that a single host does not belong to any spanning tree, because all trees around are large enough, i. e. c_h is reached. The affiliation of that host

is not possible, because tree nodes do not send messages in case the hop-counter's value is zero. When time elapses and a node does notice it does still not belong to a tree, an initiator determination is started by this host. Two cases must be distinguished. Either one the host is surrounded only by tree nodes, or several isolated hosts are existing. In both cases, the isolated host contacts all its neighbors by sending an `init`-message, and if a neighbor node already belongs to a tree it answers with a `join`-message. If no non-tree node is around, the single node chooses arbitrarily one of the neighbors and joins the tree by sending an `join-agree`-message, to the other tree hosts a `join-refuse`-message is sent. If another isolated host gets the `init`-message, a `init-agree`-message is returned and the host, sending the `init`-messagem becomes the initiator starts creating a new tree.
The obtained tree will be needed in subsection 4.5 for the global grouping process.

4.4. Local Grouping - Optimizing the Local View

In this section an algorithm is presented that determines a subset of neighbor hosts with similar profiles. Such a subset of neighbors is called a *local group*. In order to guarantee, that groups are not formed arbitrarily, but bring benefit to its members a *Profit Function* is defined. Hosts are only added to a group if the profit increases.
The algorithm adds in each step exactly one new local group member. Initially, a host scans all known profiles and selects that one, with the smallest distance to him. This profile is now tested for a profit increase. If the group with two member will bring a greater benefit, the profile is added to the local group. In this way the other profiles are either added to the local group or rejected. The local group has the shape of a line.
It may only one profile be added in a step, because else the shape of a group gets beyond control. A host H_A can add another host H_B in the exactly opposite direction than a host H_D is added by host H_C. If more than one points should be added, coordination is needed.
In order to sustain the line-shape, we only allow the two endpoints to add new points. To coordinate these two points, the endpoints of the line may add new points, alternately. If the right end has added a new point in step n, in step (n+1) the left side is on turn to add a point. The alternating procedure stops, when one side is not able to find a new point. In such a case, only the other side continues to add points, until no new point is found. Figure 3 contains the pseudocode of the algorithm.

```
firstReferenceNode := currentPoint;
secondReferenceNode := currentPoint;
nextPoint := null;
localGroup := currentPoint;
currentProfit = profit_function( localGroup );
while((nextPoint :=
     getNearestProfilePoint(firstReferenceNode))!=null)
  futureProfit:=profit_function(localGroup+nextPoint);
  if( futureProfit > currentProfit ) then
   localGroup += nextPoint;
   neighbors -= nextPoint;
   currentProfit = futureProfit;
   firstReferenceNode := secondReferenceNode;
   secondReferenceNode := nextPoint;
  fi;
elihw;
```

Figure 3. Local Grouping Algorithm

4.5. Global Grouping

In the previous section each host has identified its neighbor hosts that belong to its local group g_i. These local groups must be combined in order to achieve a global group. The algorithm presupposes no special initiator role. Each host may start the algorithm and it can even be initiated by more than one host contemporaneously. The core of the used algorithm is an echo-algorithm, see [3].
Initially, a arbitrary host sends a `EXPLORER`-message with its local-group information enclosed to its neighbors which are element of the spanning tree (the STSL, see section 4.3). If a message arrives, the enclosed local-group is taken and it is merged with its current local view of the host to get a new local view. The merging function tries to maximize the group profit function, i. e. if two groups are merged, from each group these members become a member of the new group which together draw more profit than each single group.
The new local view is forwarded to all neighbors except for the sender of the received message. If a node has no other outgoing edges and the algorithm has not terminated, the message is sent back to the sender. If more than one host initiate the algorithm and a host receives several `EXPLORER`-messages, then only the `EXPLORER`-messages from that host are forwarded, which has the highest ID (message extinction). The pseudocode of the group distribution is shown in figure 4. But if the algorithm in figure 4 has terminated, it is still not yet guaranteed, that each node has the same global view. In the worst case only the initiator node has a global view. For that reason, the echo algorithm has to be executed once more. In order to save messages, in the second run, the echo messages need not to be sent, because no further information gain is achieved.
A critical point is to determine the termination of the global grouping process. The algorithm terminates in at most $2 \cdot d = 4 \cdot c_h$ steps and because the echo messages in the second run are not sent, this is reduced to $3 \cdot c_h$. If a host receives this amount of messages, the grouping is finished.

```
Receipt of an ECHO message:
  N := N+1;
  localGroup:=merge( localGroup, ECHO.getLocalGroup() );
  if N = |STSL| then
    ENGAGED := false;
    if( initiator ) then finish;
    else
      ECHO.setLocalGroup( localGroup );
      send ECHO to PRED;
    fi;
  fi;

An EXPLORER message from host p is received:
if ( not ENGAGED )  then
 ENGAGED := true;
 N := 0;     PRED := p;
 localGroup:=getLocalGroup()
 localGroup:=merge(localGroup,EXPLORER.getLocalGroup());
 EXPLORER.setLocalGroup(localGroup);
 send EXPLORER to STSL-PRED;
fi;

N := N+1;
if ( N = |STSL| ) then
 ENGAGED := false;
 if( initiator ) then finish;
 else
  localGroup:=merge(localGroup,EXPLORER.getLocalGroup());
  ECHO.setLocalGroup(localGroup);
  send ECHO to PRED;
 fi;
fi;
```

Figure 4. Echo Algorithm and Group merging

But, due to the mobility, nodes come and go. Currently, the algorithms stops, if a node gets the same local group information from all its neighbors. This local group information is supposed to be the global group information.

4.6. Group Stability

In this subsection the stability of the groups is evaluated. With stability we mean the time a group does not change, i. e. no host is added or no group member leaves the group. This duration is very important for our algorithms, because in this stability time the group formation must be finished. We developed a simulation tool to test, how long the groups are stable. The velocity of the mobile hosts is uniformly distributed in the interval $[0; 5.2]$, the average velocity of the mobile hosts is 2.6 $\frac{km}{h}$. This speed seems to be the prevailing speed in pedestrian areas. Moreover we assume a radio transmission radius of each mobile device of 50 meters.
Figure 5 shows this dependency. It shows, that the time a group is stable, decreases rapidly with the increasing number of group members. A group with 2 people exists on the average 30 seconds, whereas a group with 5 people is only stable for 9 seconds. Nevertheless, a group that is stable for 9 seconds is still sufficient for our algorithms.

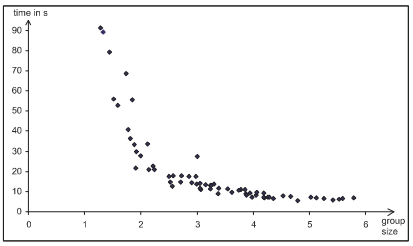

Figure 5. Group stability

5. Conclusion

In this paper we presented a kind of ad hoc applications called *Mobile Profile based Distributed Grouping* (MPDG). Each mobile host is endowed with its user's profile and while the user walks around groups are to be found, which are composed of hosts with similar profiles.
The architecture of a MPDG application is shown. The algorithm entity is responsible for local and global grouping. At first, each host has to find its local group, which consists of all neighbor hosts with similar profiles. Finally, the local groups are exchanged and a global group is achieved.
We simulated a first MPDG application, which is a taxi-sharing scenario, where potential passenger with similar destinations form a group. For the future we will apply the MPDG idea to other domains, e. g. the manufacturing or lifestyle area.

References

[1] S. Banerjee and S. Khuller. A clustering scheme for hierarchical control in multi-hop wireless networks. Technical report, University of Maryland, 2000.

[2] S. Basagni. Distributed clustering for ad hoc networks. In *Proceedings of the IEEE Int. Symposium on Parallel Architectures, Algorithms, and Networks*, 1999.

[3] E. J. H. Chang. Echo algorithms: Depth parallel operations on general graphs. *IEEE Transactions on Software Engineering*, SE-8(4):391–401, July 1982.

[4] D. Fasulo. An analysis of recent work on clustering algorithms. Technical report, University of Washington, 1999.

[5] C. Fraley and A. E. Raftery. How many clusters ? Which clustering method ? Answers via model-based cluster analysis. *The Computer Journal*, 41(8), 1998.

[6] E. M. Gafni and D. P. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, COM-29(1):11–18, January 1981.

[7] N. Malpani, J. Welch, and N. Vaidya. Leader election algorithms for mobile ad hoc networks. In *Proceedings of the Fourth International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 2000.